

Apresentação do Projeto Middleware Distribuído

Pedro Ximenes

Universidade Católica de Pernambuco (UNICAP)

22 de outubro de 2026 — 2º GQ

- 1 Contexto e Objetivo
 - Visão geral do projeto
- 2 Regras e Restrições
 - O que fazer e o que não fazer
- 3 Requisitos Obrigatórios
 - Checklist técnico
- 4 Cronograma e Entregas
 - Marcos do projeto
- 5 Rubrica de Avaliação
 - Critérios e pesos
- 6 Formação dos Grupos e Próximos Passos
 - Ações imediatas

Por que este projeto?

O cenário da indústria

Em sistemas modernos, aplicações raramente funcionam como um bloco monolítico. Elas são compostas de múltiplos serviços que precisam se comunicar de forma **confiável**, **segura** e **escalável**.

- O **middleware** é a camada invisível que torna isso possível.
- Ele gerencia: comunicação, resiliência, segurança e observabilidade entre serviços.

O papel de vocês

Cada grupo será responsável por projetar e implementar essa camada para uma aplicação real de domínio livre.

Objetivo do projeto

Enunciado

Cada grupo deve **projetar e implementar um middleware distribuído** que conecte pelo menos dois serviços de uma aplicação de domínio livre. O middleware deve ser construído do zero e demonstrar domínio prático dos conceitos de sistemas distribuídos estudados na disciplina.

Foco no middleware, não na lógica de negócio

Os serviços podem ter lógica simplificada (stubs/mocks), desde que a **comunicação entre eles** seja real e funcional.

Tecnologia

Backend em **Python, Java** ou **Go**. Infraestrutura com **Docker** (docker-compose up). Repositório no **GitHub** ou **GitLab**.

Domínios possíveis para o projeto

Exemplos de aplicação (escolha livre)

- **E-commerce:** serviço de catálogo + serviço de pedidos + pagamento.
- **Saúde:** serviço de agendamento + prontuário + notificações.
- **Logística:** rastreamento + rotas + alertas.
- **Games:** matchmaking + ranking + sessões de jogo.
- **Educação:** cursos + matrículas + avaliações.
- **IoT / monitoramento:** sensores + processamento + dashboard.
- **Qualquer domínio que o grupo domine!**

Critério de escolha

Escolha um domínio que o grupo entenda bem o suficiente para modelar as interações entre serviços. A complexidade do negócio não será avaliada.

O que o middleware deve implementar

Capacidades obrigatórias (mínimo)

Capacidade	Descrição
Comunicação síncrona	REST ou gRPC entre pelo menos 2 serviços
Comunicação assíncrona	Fila ou pub/sub em pelo menos 1 fluxo crítico
Resiliência	Pelo menos 1 mecanismo: retry, timeout, circuit breaker ou fallback
Observabilidade	Logs estruturados + métricas básicas + correlation ID
Segurança	Autenticação (token/JWT) + autorização por papéis
Tolerância a falhas	Estratégia documentada de comportamento em falhas

O que NÃO é permitido

Proibido: middleware pronto como solução principal

- **Penalidade: -3,0 pontos** na nota final.
- Exemplos proibidos como substitutos do seu middleware: Kong, Istio, MuleSoft, AWS API Gateway, Apache Camel como caixa-preta.
- Bibliotecas auxiliares são permitidas: drivers HTTP, clientes de fila, frameworks REST. O que não pode é **delegar a lógica de middleware a uma ferramenta pronta**.

Proibido: monolito disfarçado

- Serviços dentro do mesmo processo não são serviços distribuídos.
- A comunicação entre serviços deve passar pela rede (HTTP, gRPC, fila).

Proibido: repositório sem histórico

- Commits concentrados em 1–2 dias: **penalidade de -1,0 ponto**.
- O histórico de desenvolvimento é parte da avaliação.

Uso de IA/LLMs no Projeto

Permitido, com transparência obrigatória

- O uso de IA/LLMs é **liberado** para apoio em ideção, documentação, testes e código.
- A responsabilidade técnica continua sendo do grupo: vocês devem entender, revisar e validar tudo que for entregue.

Obrigatório no relatório técnico

- Indicar **onde e como** a IA/LLM foi utilizada.
- Informar **qual ferramenta/modelo** foi usado (ex.: ChatGPT, Claude, Gemini, Copilot etc.).
- Registrar os **prompts principais** usados na geração de saída relevante.

Penalidade por não declarar uso de IA/LLM

- Ausência dessa seção no relatório: **-3 pontos** na nota final.

Restrições técnicas obrigatórias

Tecnologia e infraestrutura

- **Linguagem:** Python, Java ou Go (uma linguagem principal; serviços auxiliares podem variar).
- **Containerização:** Docker obrigatório. O projeto deve subir com `docker-compose up`.
- **Repositório:** GitHub ou GitLab, público ou privado (se privado, adicionar o professor como colaborador).
- **Grupo:** até 6 integrantes.

Estrutura sugerida do repositório

- `services/` — código-fonte de cada microsserviço.
- `middleware/` — componentes centrais do middleware.
- `docs/` — relatório, diagramas, decisões.
- `docker-compose.yml` — orquestração do ambiente.
- `README.md` — instruções de execução e visão geral.

Arquitetura e Infraestrutura

Obrigatórios

- ☐ Pelo menos **2 serviços independentes** comunicando-se via middleware.
- ☐ Cada serviço em seu próprio **container Docker**.
- ☐ Projeto inteiro sobe com `docker-compose` up sem intervenção manual.
- ☐ Backend na linguagem definida: Python, Java ou Go.

Bônus

- ☐ Mais de 3 serviços com responsabilidades distintas. **(+0,2)**
- ☐ Configuração externalizada (variáveis de ambiente ou arquivo de config).

Comunicação Síncrona

Obrigatórios

- ☐ API REST ou gRPC funcional entre pelo menos 2 serviços.
- ☐ Contratos de API documentados (endpoints, payloads, códigos de resposta).

Bônus

- ☐ Implementação de gRPC com definição de .proto. (+0,2)
- ☐ Versionamento de API (v1, v2).

Comunicação Assíncrona

Obrigatórios

- ☐ Pelo menos 1 fluxo com fila de mensagens ou pub/sub (RabbitMQ, Redis Pub/Sub, NATS, Kafka...).
- ☐ Produtor e consumidor claramente separados.

Bônus

- ☐ Garantia de entrega (acknowledgment/retry na fila).
- ☐ Tratamento de mensagens duplicadas (idempotência). (+0,2)

Resiliência

Obrigatório: pelo menos 1 mecanismo

- ☐ Retry com backoff exponencial, **ou**
- ☐ Timeout configurável, **ou**
- ☐ Circuit breaker simplificado, **ou**
- ☐ Fallback (resposta degradada quando serviço está indisponível).

Bônus

- ☐ Mais de 1 mecanismo implementado. **(+0,2)**
- ☐ Teste demonstrável ao vivo: derrubar container e mostrar recuperação. **(+0,2)**

Observabilidade

Obrigatórios

- ☐ Logs estruturados (JSON ou similar) em pelo menos 1 serviço.
- ☐ Correlation ID propagado entre chamadas.
- ☐ Métricas básicas (contagem de requisições, latência, erros).

Bônus

- ☐ Dashboard de métricas (Grafana ou similar). (+0,2)
- ☐ Tracing distribuído (Jaeger, Zipkin). (+0,2)

Segurança, Tolerância a Falhas e Repositório

Segurança — obrigatórios

- ☐ Autenticação em pelo menos 1 endpoint (token simples, JWT ou API key).
- ☐ Autorização por papéis (role-based) em pelo menos 1 recurso.

Tolerância a Falhas — obrigatórios

- ☐ Documento explicando a estratégia de consistência escolhida (eventual, forte, etc.).
- ☐ Comportamento definido e documentado para quando um serviço cai.

Repositório e Documentação — obrigatórios

- ☐ Histórico de commits distribuído ao longo do projeto.
- ☐ README com instruções claras de execução.
- ☐ Relatório técnico (3–6 páginas): arquitetura, decisões, diagramas e lições.
- ☐ Diagrama de arquitetura atualizado.

Cronograma resumido

Linha do tempo — 2º GQ

Data	Marco / Atividade
22/10	Apresentação do projeto (esta aula)
26/10	Aula Prática: Containers, Docker e grupos
29/10	Mensageria e Comunicação Assíncrona
05/11	Tolerância a Falhas em Sistemas Distribuídos
09/11	Checkpoint — Prática: MVP do Middleware e Métricas
12/11	Segurança e Resiliência em Sistemas Distribuídos
16/11	Laboratório de métricas e avaliação
19/11	Apresentações do Projeto — Fase 1
23/11	Apresentações do Projeto — Fase 2
30/11	Aplicação da Prova do 2º GQ

Checkpoint — 09/nov: o que ter pronto

Acompanhamento intermediário (Prática de 09/nov)

- MVP funcional do middleware rodando em Docker.
- Pelo menos comunicação síncrona e assíncrona funcionando.
- Início da extração de métricas de sistema.
- Diagrama atualizado da arquitetura.

Checklist do checkpoint

- ☐ Repositório atualizado com commits recentes.
- ☐ Arquitetura revisada (fluxo síncrono + assíncrono no diagrama).
- ☐ Contratos de serviço minimamente definidos (REST/gRPC + eventos).
- ☐ `docker compose up` funcionando para os serviços principais.
- ☐ Lista de pendências priorizada para a entrega final.
- ☐ Plano de segurança, resiliência e observabilidade (mesmo que parcial).

Entrega final e apresentações — 19/nov e 23/nov

Componentes da entrega

Componente	Descrição
Demo ao vivo	Demonstração funcional do middleware e serviços rodando
Apresentação (slides)	10–15 min explicando decisões arquiteturais
Repositório	Código organizado, README, docker compose funcional
Relatório técnico	3–6 páginas: arquitetura, decisões, diagramas, dificuldades

Estrutura sugerida da apresentação

- 1 **Contexto** (1–2 min): qual o domínio e por que o escolheram.
- 2 **Arquitetura** (3–4 min): diagrama, serviços, middleware, decisões.
- 3 **Demo ao vivo** (4–5 min): fluxos principais funcionando.
- 4 **Desafios e lições** (2–3 min): o que deu errado, o que aprenderam.
- 5 **Perguntas** (5 min): professor e turma.

Visão geral da rubrica

Nota máxima: 10,0 pontos

Critério	Peso
Arquitetura e Middleware	3,0
Apresentação e Demo	3,0
Repositório e Documentação	2,0
Relatório Técnico	2,0
Itens Bônus	+1,0
Total máximo	10,0

Atenção

Cada aspecto da rubrica é avaliado como: completo (valor total), parcial (metade) ou ausente (zero).

Rubrica: Arquitetura e Middleware (3,0 pts)

Detalhamento por aspecto

Aspecto	Pts	Descrição
Comunicação síncrona	0,5	REST ou gRPC funcional entre $\geq$ 2 serviços
Comunicação assíncrona	0,5	Fila/pub-sub com produtor e consumidor separados
Resiliência	0,5	$\geq$ 1 mecanismo implementado
Observabilidade	0,5	Logs + correlation ID + métricas básicas
Segurança	0,5	Autenticação + autorização por papéis
Tolerância a falhas	0,5	Estratégia documentada e comportamento em falhas

Rubrica: Apresentação e Demo (2,5 pts)

Critérios de avaliação da apresentação

Aspecto	Pts
Demo funcional (docker-compose up + fluxos)	1,0
Clareza da apresentação (decisões arquiteturais)	0,5
Domínio do conteúdo (todos os integrantes)	0,5
Qualidade dos slides (diagramas, sem excesso de texto)	0,25
Gestão do tempo (10–15 min)	0,25

Rubrica: Repositório e Relatório

Repositório e Doc. (2,0 pts)

- Organização do código — 0,5
- README funcional — 0,5
- Docker funcional — 0,5
- Histórico de commits — 0,5

Relatório Técnico (1,5 pts)

- Descrição da arquitetura — 0,5
- Decisões técnicas — 0,5
- Dificuldades e lições — 0,5

Penalidades

Atenção: situações que geram desconto

Situação	Pen.
Middleware pronto como solução (Kong, Istio, etc.)	-3,0
Repositório sem histórico de desenvolvimento	-1,0
Projeto não sobe com docker compose up	-1,0
Uso de IA/LLM sem declaração no relatório técnico	-3,0
Integrante ausente na apresentação (sem justif.)	-0,5
Atraso na entrega (por dia)	-0,5
Plágio ou cópia de outro grupo	Nota zero

Formação dos grupos

Regras

- Grupos de até **5 a 6 integrantes**.
- Formação até **26/10** (Aula Prática de Containers e Docker).
- Cada grupo deve comunicar ao professor:
 - Nome do grupo.
 - Integrantes (nome + matrícula).
 - Domínio escolhido.
 - Link do repositório (GitHub/GitLab) já criado.

Divisão de responsabilidades sugerida (grupo de 5)

- 1 pessoa por serviço/componente do middleware.
- 1 pessoa responsável por Docker/infraestrutura.
- Rotação nas revisões de código (pull requests).

Checklist para 26/10 — Aula Prática

O que o grupo deve definir para a aula prática

- ☐ Grupo formado e comunicado ao professor.
- ☐ Domínio da aplicação escolhido.
- ☐ Repositório criado no GitHub/GitLab.
- ☐ Professor adicionado como colaborador.
- ☐ Diagrama inicial de arquitetura (serviços + middleware, mesmo que esboço).
- ☐ `docker compose up` subindo pelo menos 2 containers.
- ☐ README com nome do grupo, domínio e instruções básicas.

Atenção: não precisa estar funcional ainda

Em 26/10, o esperado é: grupo definido, repositório criado e containers básicos subindo. A lógica do middleware será desenvolvida ao longo das próximas semanas.

Dicas para um projeto bem-sucedido

Estratégia de desenvolvimento

- **Comece simples:** 2 serviços + REST + Docker já é um bom ponto de partida. Adicione complexidade incrementalmente.
- **Documente desde o início:** diagramas e decisões técnicas facilitam o relatório e a apresentação.
- **Dividam responsabilidades:** não deixem tudo para uma pessoa. O histórico de commits reflete a participação individual.
- **Testem a resiliência:** derrubem um container e vejam o que acontece. O middleware deve lidar com isso graciosamente.
- **Não reinventem a roda:** usar biblioteca HTTP ou driver de fila é esperado. O que vocês devem construir é a **lógica de orquestração**.

Critério de sucesso simples

Se vocês conseguem rodar `docker compose up`, derrubar um serviço e o sistema continuar respondendo (degradado ou com `retry`), o núcleo do projeto está feito.